Designing of Optimized Critical Path Delay in MCM using Truncation Logic Multiplication

Kaushiki Sharma[1], Prof. Ashish Raghuwanshi[2]
[1]*M-Tech Research Scholar,* [2]*Research Guide*
Department of Electronics & Communication Engineering, IES, Bhopal

Abstract - Multiplying by known constants is a common operation in many digital signal processing (DSP) algorithms. High performance DSP systems are implemented in custom hardware, in which the designer has the ability to choose which logic elements will be used to perform the computation. By exploiting the properties of binary multiplication, it is possible to realize constant multiplication with fewer logic resources than required by a generic multiplier. This work proposed an architecture which is more efficient in terms of area as well as delay than the existing system. The critical path analysis has been done using architecture optimization of the system utilizing booth multiplier. This work proposed an architecture which is more efficient in terms of area as well as delay than the existing system. The critical path analysis has been done using architecture optimization of the system utilizing booth multiplier. The proposed design significantly improve the delay upto 3.483ns which is 26.06% less than existing system.

Keywords - FPGA, Virtex 7, Delay Efficient, MCM, Critical Path Analysis.

I. INTRODUCTION

Multiplying a variable by a set of known constant coefficients is a common operation in many digital signal processing (DSP) algorithms. Compared to other common operations in DSP algorithms, such as addition, subtraction, using delay elements, etc., multiplication is generally the most expensive. There is a trade-off between the amount of logic resources used (i.e. the amount of silicon in the integrated circuit) and how fast the computation can be done. Compared to most of the other operations, multiplication requires more time given the same amount of logic resources and it requires more logic resources under the constraint that each operation must be completed within the same amount of time.

A general multiplier is needed if one performs multiplication between two arbitrary variables. However, when multiplying by a *known constant*, we can exploit the properties of binary multiplication in order to obtain a less expensive logic circuit that is functionally equivalent to simply asserting the constant on one input of a general multiplier. In many cases, using a cheaper implementation for only multiplication still results in significant savings when considering the entire logic circuit because multiplication is relatively expensive. Furthermore, multiplication could be the dominant operation, depending on the application.

In this thesis, we will propose several algorithms which *run in software*, but the solutions that these algorithms produce enable one to efficiently implement constant coefficient multiplication in *hardware. Given* the set of constant coefficients, said algorithms *search* for good hardware realizations.

Historically, ECL has been the choice when the highest speed was desired, it's main drawback being high power consumption. Although CMOS has been closing the speed gap, at high speeds it too is a high power technology. At the present time ECL, as measured by loaded gate delays, is somewhere between \ and \ the delay of similar CMOS gates. Comparable designs in ECL also take about the same layout area as a CMOS design, primarily because the metal interconnect limits the circuit densities. Because ECL seems to still maintain a speed advantage, the technology used as a basis for this work will be ECL, supplemented with differential ECL where possible. Most conclusions will apply primarily to implementations using ECL, but wherever possible, the results will be generalized to other implementation technologies, principally CMOS.

II. MULTIPLICATION ARCHITECTURES

Chapter 3 presents partial product generation in detail, but all multiplication methods share the same basic procedure - addition of a number of partial products. A number of different methods can be used to add the partial products. The simple methods are easy to implement, but the more complex methods are needed to obtain the fastest possible speed.

Iterative

The simplest method of adding a series of partial products is shown in Figure 2.1. It is based upon an adder-accumulator, along with a partial product generator and a hard wired shifter. This is relatively slow, because adding N partial products requires N clock cycles. The easiest clocking scheme is to make use of the system clock, if the multiplier is embedded in a larger system. The system clock is normally much slower than the maximum speed at which the simple iterative multiplier can be clocked, so if the delay is to be minimized an expensive and tricky clock

multiplier is needed, or the hardware must be self-clocking.

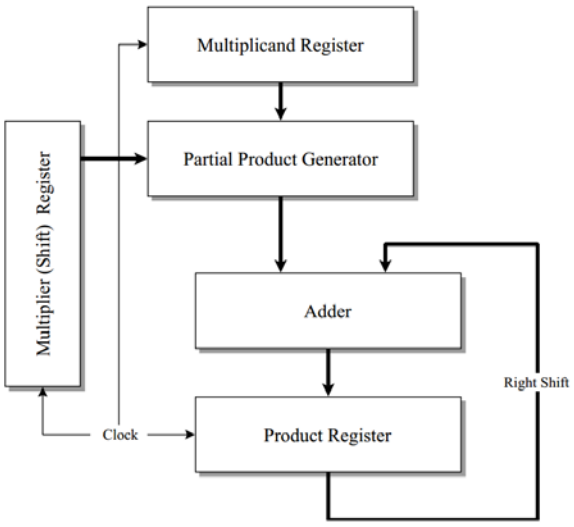

Figure 2.1: Simple iterative multiplier.

Linear Arrays

A faster version of the basic iterative multiplier adds more than one operand per clock cycle by having multiple adders and partial product generators connected in series (Figure 2.2). This is the equivalent of "unrolling" the simple iterative method. The degree to which the loop is unrolled determines the number of partial products that can be reduced in each clock cycle, but also increases the hardware requirements. Typically, the loop is unrolled only to the point where the system clock matches the clocking rate of this multiplier. Alternately, the loop can be unrolled completely, producing a completely combinatorial multiplier (a full linear array). When contrasted with the simple iterative scheme, it will match the system clock speed better, making the clocking much simpler. There is also less overhead associated with clock skew and register delay per partial product reduced.

Parallel Addition (TREES)

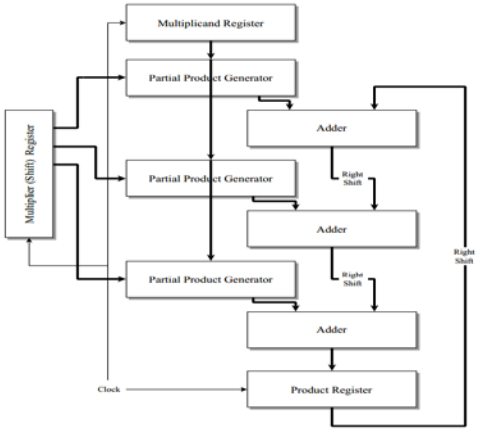

Figure 2.2: Linear array multiplier.

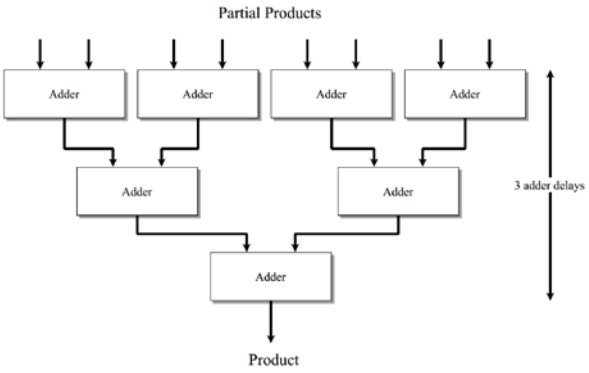

Figure 2.3: Adding 8 partial products in parallel.

When a number of partial products are to be added, the adders need not be connected in series, but instead can be connected to maximize parallelism, as shown in Figure 2.3. This requires no more hardware than a linear array, but does have more complex interconnections. The time required to add N partial products is now proportional to log N, so this can be much faster for larger values of N. On the down side, the extra complexity in the interconnection of the adders may contribute to additional size and delay.

Wallace Trees

The performance of the above schemes are limited by the time to do a carry propagate addition. Carry propagate adds are relatively slow, because of the long wires needed to propagate carries from low order bits to high order bits. Probably the single most important advance in improving the speed of multipliers, pioneered by Wallace [5], is the use of carry save adders (CSAs also known as full adders or 3-2 counters [7]), to add three or more numbers in a redundant and carry propagate free manner. The method is illustrated in Figure 2.4. By applying the basic three input adder in a recursive manner, any number of partial products can be added and reduced to 2 numbers without a carry propagate adder. A single carry propagate addition is only needed in the final step to reduce the 2 numbers to a single, final product. The general method can be applied to trees and linear arrays alike to improve the performance.

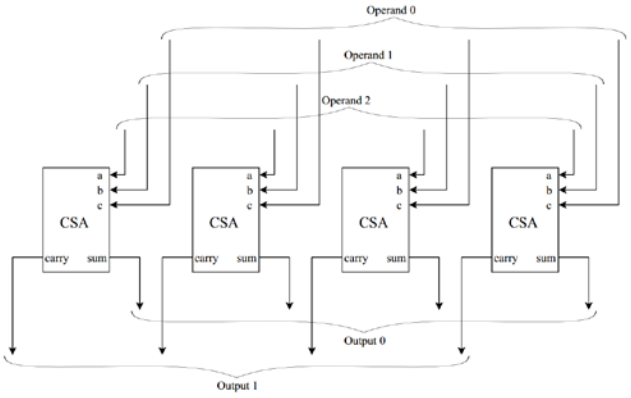

Figure 2.4: Reducing 3 operands to 2 using CSAs.

Binary Trees

The tree structure described by Wallace suffers from irregular interconnections and is difficult to layout. A more regular tree structure is described by [7], all of which are based upon binary trees. A binary tree can be constructed by using a row of 4-2 counters, which accepts 4 numbers and sums them to produce 2 numbers. Although this improves the layout problem, there are still irregularities, an example of which is shown in Figure. This figure shows the reduction of 8 partial products in two levels of 4-2 counters to two numbers, which would then be reduced to a final product by a carry propagate adder. The shifting of the partial products introduce zeros at various places in the reduction. These zeros represent either hardware inefficiency, if the zeros are actually added, or irregularities in the tree if special counters are built to explicitly exclude the zeros from the summation. The figure shows bits that jump levels (gray dots), and more counters in the row making up the second level of counters (12), than there are in the rows making up the first level of counters (9). All of these effects contribute to irregularities in the layout, although it is still more regular than a Wallace tree.

Applications of Constant Multiplication

In this section, we will provide a few examples of applications which require multiplication by a set of constants. When *designing the hardware* to implement these applications, the algorithms in this thesis can be used to provide good solutions for the constant coefficient multiplication part of the logic circuit.

Multiplication by a set of constants occurs when multiplying by a constant vector or a constant matrix. For example, the dot product $a \times b$ gives the scalar projection of a onto b (or vice versa). Multiplication by a constant matrix is nothing more than performing the dot product between several constant vectors (which collectively form a matrix) and a variable vector (the elements of this vector are the inputs). Multiplying by a constant matrix can thus be regarded as a linear transformation of coordinates, which is used in many applications. For example, the conversion from the RGB (red, green, blue) color space to the YUV color space (Y represents brightness, U and V represent chroma) involves multiplication by a constant 3x3 matrix. Because the human eye is more sensitive to brightness than coloring (chroma), we can compress the information in the U and V components with only a minor perceived distortion.

III. PROPOSED ARCHITECTURE

The system proposed for the critical path analysis of multiple constant multiplication (MCM) is shown in the below figure. The proposed model utilizes the booth multiplication logic to execute the whole system which are displayed as truncated multiplier blocks in different four stages.

Table 1: Timing Details

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)
=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 3519 / 8
-------------------------------------------------------------------------
Delay:               3.483ns (Levels of Logic = 11)
  Source:            b<7> (PAD)
  Destination:       yy<7> (PAD)
  Data Path: b<7> to yy<7>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             13   0.003   0.614  b_7_IBUF (b_7_IBUF)
     LUT6:I0->O             2   0.040   0.471  M22/a12/Mxor_su_xo<0>1 (x2<4>)
     LUT4:I0->O             2   0.040   0.563  M33/m6/ca1 (y1<3>)
     LUT6:I0->O             2   0.040   0.564  M44/n4/ca1 (z1<4>)
     LUT6:I1->O             1   0.040   0.349  M55/f4/Mxor_su_xo<0>1 (M55/w<4>)
     LUT2:I0->O             1   0.040   0.000  M55/Madd_c_lut<4> (M55/Madd_c_lut<4>)
     MUXCY:S->O             1   0.223   0.000  M55/Madd_c_cy<4> (M55/Madd_c_cy<4>)
     MUXCY:CI->O            1   0.012   0.000  M55/Madd_c_cy<5> (M55/Madd_c_cy<5>)
     MUXCY:CI->O            0   0.012   0.000  M55/Madd_c_cy<6> (M55/Madd_c_cy<6>)
     XORCY:CI->O            1   0.189   0.279  M55/Madd_c_xor<7> (yy_7_OBUF)
     OBUF:I->O                  0.002          yy_7_OBUF (yy<7>)
    ----------------------------------------
    Total                       3.483ns (0.642ns logic, 2.841ns route)
                                        (18.4% logic, 81.6% route)
=========================================================================
```

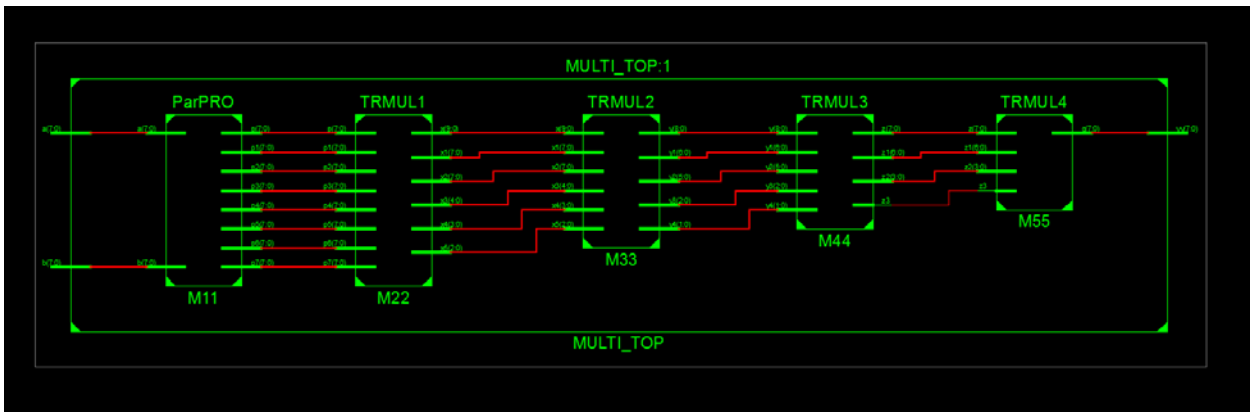

Figure 3.1 Architecture of Proposed Method

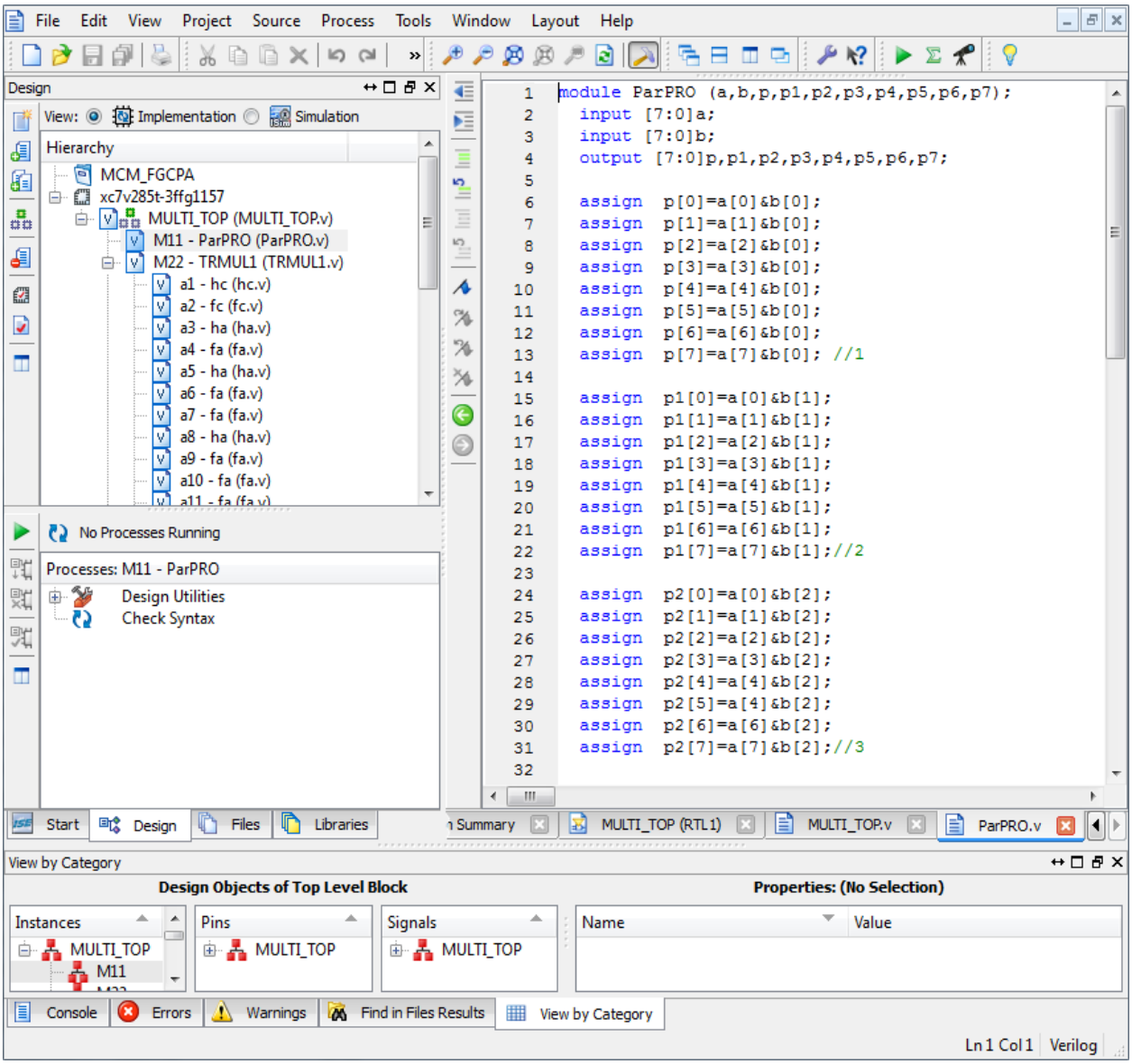

Figure 3.2 Screen Shot of the XILINX Synthesis

Table 2: Comparison of Parameters

Architecture	Delay	Area Utilization
Proposed Using Booth Multiplier	3.483 ns	0.009%
Existing using GA	14.3 ns	2.9%

Table 3: Device Utilization Summary

```
Device utilization summary:
---------------------------
Selected Device : 7v285tffg1157-3

Slice Logic Utilization:
 Number of Slice LUTs:                  62  out of  178800     0%
    Number used as Logic:               62  out of  178800     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    62
   Number with an unused Flip Flop:     62  out of     62   100%
   Number with an unused LUT:            0  out of     62     0%
   Number of fully used LUT-FF pairs:    0  out of     62     0%
   Number of unique control sets:        0

IO Utilization:
 Number of IOs:                         24
 Number of bonded IOBs:                 24  out of    600     4%
```

IV. CONCLUSION AND FUTURE SCOPE

The architecture of the proposed model for lower delay critical path analysis of multiple constant multiplication (MCM) utilizing booth multiplier is discussed in the previous section. The area is also calculated which is also lower. The CPD analysis shows the system is better than the existing in terms of complexity, delay and area. The comparison is also shown in the Table 2 and the details of delay and area utilization which will also shown in the Table 1 and Table 3 respectively.

REFERENCES

[1] X. Lou, Y. J. Yu and P. K. Meher, "Fine-Grained Critical Path Analysis and Optimization for Area-Time Efficient Realization of Multiple Constant Multiplications," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 62, no. 3, pp. 863-872, March 2015.

[2] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde and D. Durackova, "A new algorithm for elimination of common subexpressions," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 1, pp. 58-68, Jan 1999.

[3] M. Martinez-Peiro, E. I. Boemo and L. Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm," in IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 49, no. 3, pp. 196-203, Mar 2002.

[4] Fei Xu, Chip-Hong Chang and Ching-Chuen Jong, "Contention resolution algorithm for common subexpression elimination in digital filter design," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 52, no. 10, pp. 695-700, Oct. 2005.

[5] O. Gustafsson and L. Wanhammar, "ILP modelling of the common subexpression sharing problem," Electronics, Circuits and Systems, 2002. 9th International Conference on, 2002, pp. 1171-1174 vol.3.

[6] L. Aksoy, E. da Costa, P. Flores and J. Monteiro, "Exact and Approximate Algorithms for the Optimization of Area and Delay in Multiple Constant Multiplications," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 27, no. 6, pp. 1013-1026, June 2008.

[7] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," in IEE Proceedings G - Circuits, Devices and Systems, vol. 138, no. 3, pp. 401-412, June 1991.

[8] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," in IEE Proceedings - Circuits, Devices and Systems, vol. 141, no. 5, pp. 407-413, Oct 1994.

[9] O. Gustafsson, A. G. Dempster and L. Wanhammar, "Extended results for minimum-adder constant integer multipliers," Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium on, 2002, pp. I-73-I-76 vol.1.